บุญแสง สุรวัฒนบุตร

ยูทิลิตี้ที่ช่วยในการ ชำแหละดิสค์

ผ่าโครงสร้างภายในดิสค์ แหล่งเก็บข้อมูล ที่ใช้งานกันเป็นประจำ ด้วยโปรแกรมยูทิลิตี้ เพื่อแก้ปัญหาด้วยตัวเอง

สิ่งที่ผู้ใช้เครื่องไมโครคอมพิวเตอร์ทุกคนจะต้องพบเป็นประจำคือแผ่นดิสค์ เราใช้แผ่นดิสค์ในการเก็บโปรแกรมและข้อมูลต่าง ๆ โดยไม่สนใจลักษณะการจัดเก็บข้อมูลบนดิสค์เท่าใดนัก เพราะดอสจะคอยทำหน้าที่ดังกล่าว แต่บางครั้งความผิดพลาดย่อมเกิดขึ้นได้ถ้าวันหนึ่งเราใช้คำสั่ง DIR บนดอสแล้วปรากฏว่าไม่สามารถอ่านไดเรคทอรีได้ แต่ข้อมูลที่จัดเก็บยังอยู่บนแผ่นดิสค์ หากเราทราบลักษณะการเก็บข้อมูลบนแผ่นดิสค์และข้อมูลมีความสำคัญที่ต้องเก็บรักษาไว้ การกู้ข้อมูลด้วยตัวเราเองก็มีทางเป็นไปได้ บทความต่อไปนี้จะกล่าวถึงการใช้อินเตอร์รัพต์ของรอมไบออสในการอ่านข้อมูลที่อยู่ในเซกเตอร์บนดิสค์ เพื่อนำมาพิจารณาลักษณะการเก็บข้อมูลของดิสค์ ซึ่งจะพิจารณาจากแผ่นดิสค์ที่นิยมใช้กันทั่วไปก่อนคือ แผ่นดิสค์แบบสองหน้าและความหนาแน่นสองเท่า (DS,DD) โดยเขียนโปรแกรมตัวอย่างด้วยเทอร์โบปาสคาล

มีอะไรบ้างในแผ่นดิสค์

ก่อนที่จะทำการศึกษาการเก็บข้อมูลบนแผ่นดิสค์ เราควรทราบถึงส่วนประกอบต่าง ๆ ของแผ่นดิสค์เสียก่อนดังแสดงในรูปที่ 1

– Write-Protect Notch เป็นตัวกำหนดการอนุญาตในการเขียนหรือการแก้ไขข้อมูลบนแผ่นดิสค์ หากไม่ได้ติด Write Protect เราจะสามารถเขียนหรือทำการแก้ไขข้อมูลบนแผ่นได้ แต่ถ้าติด Write-Protect ไว้ จะอ่านข้อมูลบนดิสค์ได้แต่แก้ไขไม่ได้ ถ้าการเขียนข้อมูลบนแผ่นที่ติด ดอสจะแจ้งให้ทราบว่า Write protect error writing drive X: เป็นต้น (X:คือไดรฟ์ที่ทำการเขียนข้อมูล)

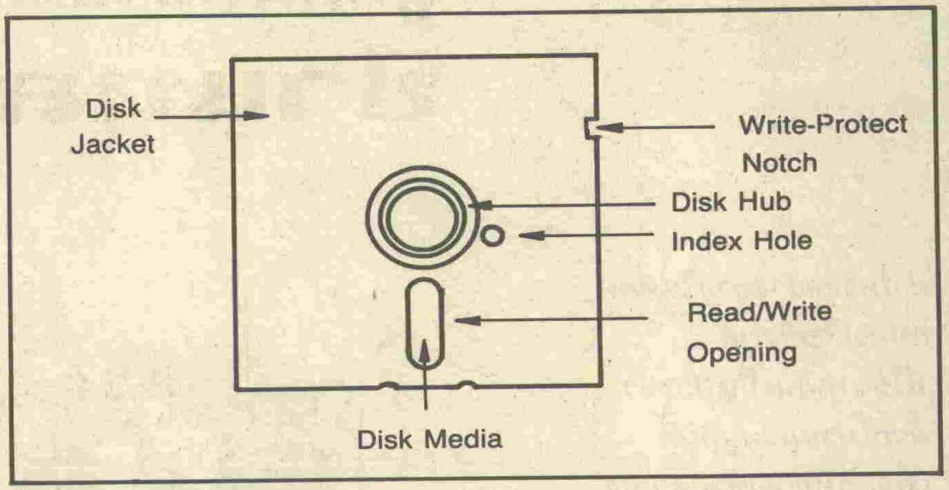
– Disk Media หมายถึงเนื้อดิสค์ที่ใช้ในการเก็บรักษาข้อมูล

– Disk Jacket เป็นส่วนที่หุ้มเนื้อดิสค์ทำหน้าที่ป้องกันฝุ่นละออง เขม่าควัน ลายนิ้วมือจากการจับแผ่นดิสค์ และยังสามารถติดป้ายแสดงรายละเอียดต่าง ๆ ของข้อมูลหรือโปรแกรมที่อยู่บนแผ่นดิสค์อีกด้วย

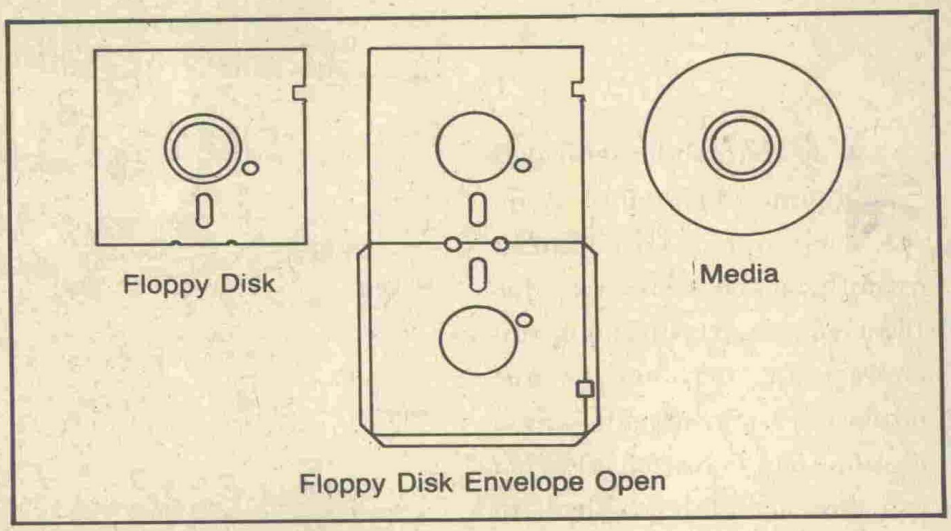
– Read/Write Opening เป็นช่องเปิดของ Disk Jacket เพื่อให้หัวอ่าน/เขียนของไดรฟ์สามารถอ่าน/เขียนข้อมูลจากแผ่นดิสค์ได้

– Disk Hub เป็นส่วนที่ไดรฟ์ใช้ในการหมุนแผ่นดิสค์ขณะอ่านหรือเขียนข้อมูล เนื่องจากตำแหน่งของข้อมูลบนแผ่นดิสค์อาจอยู่ในตำแหน่งที่ไม่ตรงกับ Read/Write Opening ดังนั้นในการอ่านหรือเขียนข้อมูลลงบนดิสค์จึงต้องหมุนแผ่นดิสค์ เพื่อให้ข้อมูลที่ต้องการตรงกับหัวอ่าน/เขียนของไดรฟ์ผ่าน Read/Write Opening

– Index Hole เป็นส่วนที่ช่วยควบคุมการทำงานที่เกี่ยวกับแผ่นดิสค์ รูของ Index Hole จะอยู่บนเนื้อดิสค์และมีช่องเปิดบน Disk Jacket ด้วย เป็นตำแหน่งสัมพัทธ์เพื่อให้สามารถรู้ตำแหน่งอื่น ๆ บนดิสค์ได้ เปรียบเสมือนเครื่องหมายบอกตำแหน่งเริ่มต้นเพื่อใช้ในการหาตำแหน่งอื่น จากลักษณะภายนอกของแผ่นดิสค์ที่แสดงในรูปที่ 1 เมื่อคลี่ Jacket ออกจะเห็นลักษณะภายในดังรูปที่ 2 จะเห็นว่าตัวเนื้อดิสค์มีลักษณะกลมเพื่อให้หมุนได้



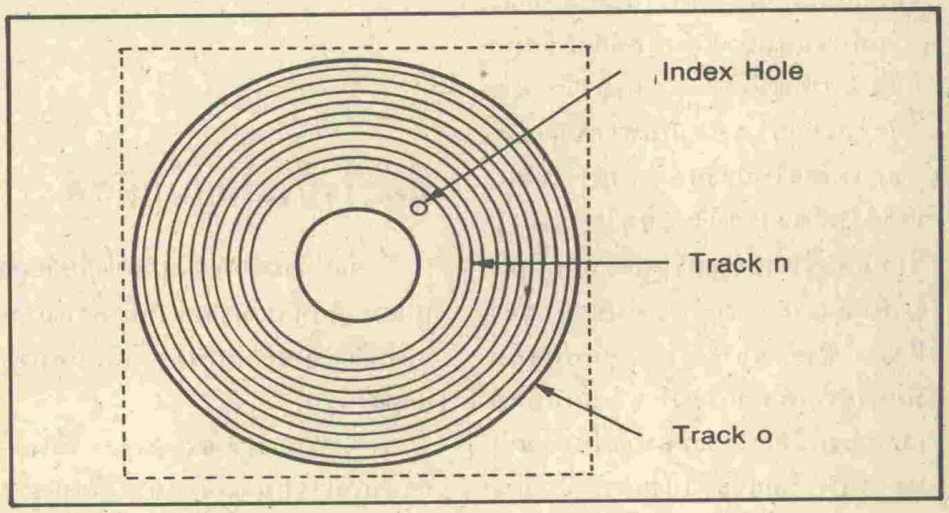
รูปที่ 1 แสดงส่วนประกอบของดิสค์



รูปที่ 2 แสดงลักษณะภายในของดิสค์

จัดการแบ่งพื้นที่บนดิสค์

การแบ่งพื้นที่บนดิสค์จะแบ่งเป็นชั้นรอบ ๆ ดิสค์ แต่ละชั้นเรียกว่าแทร็ก (track) โดยเริ่มจากแทร็ก 0 ที่ขอบนอกเรียงตามลำดับดังรูปที่ 3 และในแต่ละแทร็กยังแบ่งออกเป็นเซกเตอร์ (sector)

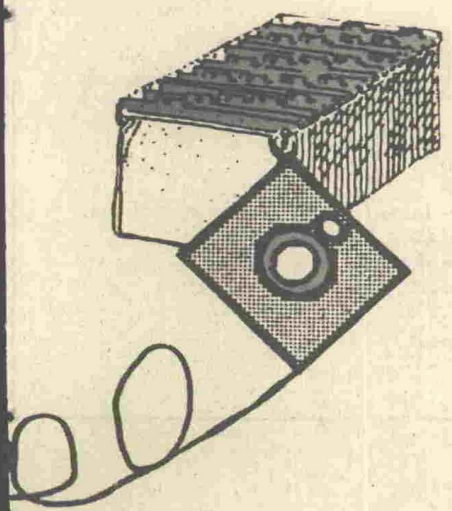


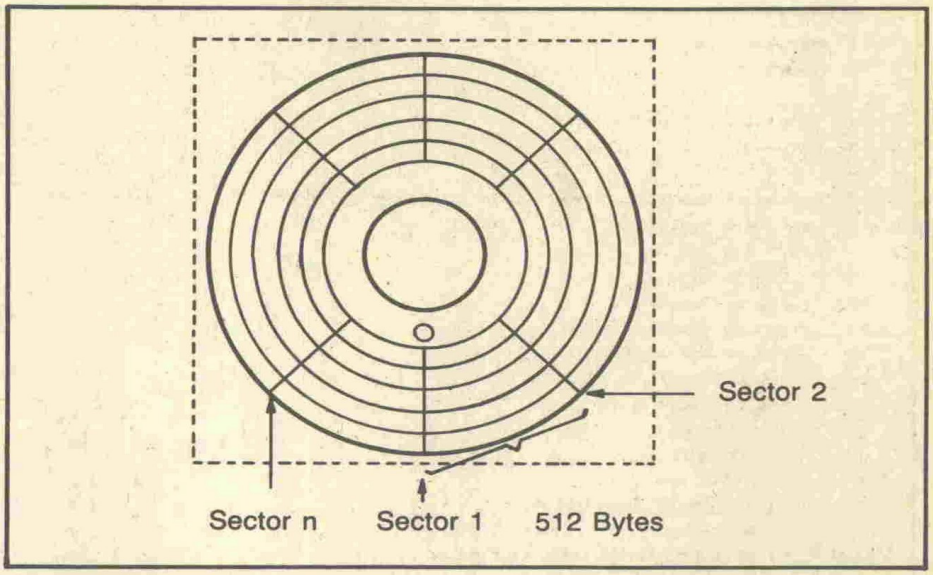
รูปที่ 3 แสดงการแบ่งพื้นที่บนดิสค์

เริ่มจากเซกเตอร์ที่ 1 คือตำแหน่งของ Index Hole ดังรูปที่ 4 จำนวนของแทร็กและเซกเตอร์ขึ้นอยู่กับประเภทของแผ่นดิสค์ จากตารางที่ 1 แสดงจำนวนแทร็กต่อหน้าและจำนวนเซกเตอร์ทั้งหมดบนดิสค์แต่ละประเภท จะพบว่าแผ่นดิสค์ทั่วไป (DS,DD) มี 9 เซกเตอร์ต่อแทร็ก และมีจำนวนแทร็กเท่ากับ 40 แทร็กต่อหน้า โดยเริ่มจากแทร็ก 0 จนถึงแทร็ก 39 แผ่นดิสค์ที่ใช้กันทั่วไปสามารถจุข้อมูลได้ 512 ไบต์ต่อเซกเตอร์ สามารถจุข้อมูลได้ทั้งหมดเท่ากับ

= จำนวนแทร็กต่อหน้า × จำนวนเซกเตอร์ต่อแทร็ก × จำนวนหน้า × 512
= 40 × 9 × 2 × 512
= 368640 ไบต์

ในบทความนี้จะใช้วิธีบอกตำแหน่งบนดิสค์ (หมายเลขแทร็ก, หมายเลขหน้า, หมายเลขเซกเตอร์) เช่น (20,0,4) หมายถึงตำแหน่งแทร็กที่ 20 หน้าที่ 0 เซกเตอร์ที่ 4 นอกจากการกำหนดตำแหน่งบนแผ่นดิสค์ด้วยการกำหนดหมายเลขแทร็ก, หมายเลขเซกเตอร์ และหมายเลขหน้าแล้วยังสามารถบอกตำแหน่งบนดิสค์ด้วยหมายเลขเซกเตอร์สมบูรณ์ (absolute sector) โดยเริ่มจาก





รูปที่ 4 แสดงพื้นที่ในการเก็บข้อมูลบนดิสค์

ประเภทของดิสค์	จำนวนแทร็กต่อหน้า	จำนวนเซกเตอร์
หน้าเดียว 8 เซกเตอร์ต่อแทร็ก	40	320
หน้าเดียว 9 เซกเตอร์ต่อแทร็ก	40	360
สองหน้า 8 เซกเตอร์ต่อแทร็ก	40	640
สองหน้า 9 เซกเตอร์ต่อแทร็ก	40	720
หนาแน่น 4 เท่า 9 เซกเตอร์ต่อแทร็ก	80	1440
หนาแน่น 4 เท่า 15 เซกเตอร์ต่อแทร็ก	80	2400

ตารางที่ 1 แสดงจำนวนแทร็กต่อหน้าและจำนวนเซกเตอร์ทั้งหมดบนดิสค์แต่ละประเภท

เซกเตอร์สัมบูรณ์ที่	0	ตรงกับตำแหน่ง (0.0.1)
ถึงเซกเตอร์สัมบูรณ์ที่	8	ตรงกับตำแหน่ง (0.0.9)
ไปต่อเซกเตอร์สัมบูรณ์ที่	9	ที่ตำแหน่ง (0.1.1)
เซกเตอร์สัมบูรณ์ที่	18	ตรงกับตำแหน่ง (1.0.1)
เซกเตอร์สัมบูรณ์ที่	27	ตรงกับตำแหน่ง (1.1.1)
เซกเตอร์สัมบูรณ์ที่	719	ตรงกับตำแหน่ง (39.1.9)

การเก็บข้อมูลลงบนแผ่นดิสค์ ดอสจะเก็บหรืออ่านข้อมูลเป็นกลุ่มเรียกว่า คลัสเตอร์ สำหรับแผ่นดิสค์ทั่วไปมีขนาด 2 เซกเตอร์ต่อคลัสเตอร์ เริ่มต้นจาก

คลัสเตอร์หมายเลข	2	ที่เซกเตอร์สัมบูรณ์หมายเลข	12
คลัสเตอร์หมายเลข	3	ที่เซกเตอร์สัมบูรณ์หมายเลข	14
คลัสเตอร์หมายเลข	4	ที่เซกเตอร์สัมบูรณ์หมายเลข	16
คลัสเตอร์หมายเลข	355	ที่เซกเตอร์สัมบูรณ์หมายเลข	718

ถ้าข้อมูลของไฟล์ที่เก็บไม่เต็มคลัสเตอร์ ส่วนที่เหลือในคลัสเตอร์ถือว่าเป็น ส่วนหนึ่งของไฟล์ ๆ นั้น ไฟล์อื่นไม่สามารถนำไปใช้ประโยชน์ได้ ขนาดที่แท้จริงของไฟล์จะเก็บไว้ในไดเรคทอรีเพื่อใช้ในการบอกตำแหน่งสิ้นสุดของไฟล์

วิธีอ่านข้อมูลในเซกเตอร์

เราจะใช้อินเตอร์รัพต์หมายเลข $13 ฟังก์ชัน 2 ในการอ่านข้อมูลจากเซกเตอร์บนแผ่นดิสค์ รูปที่ 5 เป็นการส่งค่ารีจิสเตอร์ ค่าของ DL หมายถึงไดรฟ์ที่ต้องการอ่านข้อมูล โดยไดรฟ์

```
 1: PROGRAM DISK;
 2: USES Crt,Dos;
 3:
 4: TYPE  BufferType  = ^char;
 5: VAR   ch          : char;
 6:
 7: Procedure ReadSector(drv,trck,sctr,sd,nsctr:integer;Buffer:BufferType);
 8: var reg : Registers;
 9: begin
10:    reg.ah:=02;
11:    reg.al:=nsctr;
12:    reg.es:=Seg(Buffer^);
13:    reg.bx:=Ofs(Buffer^);
14:    reg.ch:=trck;
15:    reg.cl:=sctr;
16:    reg.dh:=sd;
17:    reg.dl:=drv;
18:    intr($13,reg);
19:    if reg.ah<>0 then begin
20:       WriteLn;
21:       WriteLn('ERROR CODE =',reg.ah);
22:       Halt(1);
23:    end;
24: end;
25:
```

รูปที่ 5 โปรแกรม DISK.PAS

Call with:	AH	= 02H
	AL	= จำนวนเซกเตอร์ที่จะอ่าน
	ES:BX	= เซกเมนต์และออฟเซ็ตของบัฟเฟอร์ที่กำหนด
	CH	= หมายเลขแทร็ก (0-39)
	CL	= หมายเลขเซกเตอร์ (1-9)
	DH	= หมายเลขหน้าของดิสค์ (0-9)
	DL	= หมายเลขไดร์ฟ (0-3)
Return:	ถ้าการอ่านสำเร็จ	
	Carry flag	= clear
	AH	= 0
	AL	= จำนวนเซกเตอร์ที่อ่านได้
	ถ้าการอ่านล้มเหลว	
	Carry flag	= set
	AH	= บอกสถานะของข้อผิดพลาดถูกเซตเมื่อ
		บิท 0 = illegal command passed to drive
		บิท 1 = disk write-protected
		บิท 2 = requested sector not found
		บิท 3 = DMA overrun on operation
		บิท 4 = data error on disk read (CRC)
		บิท 5 = controller error
		บิท 6 = seek failure
		บิท 7 = disk timed-out (failed to respond)

รูปที่ 6 แสดงการส่งค่ารีจิสเตอร์

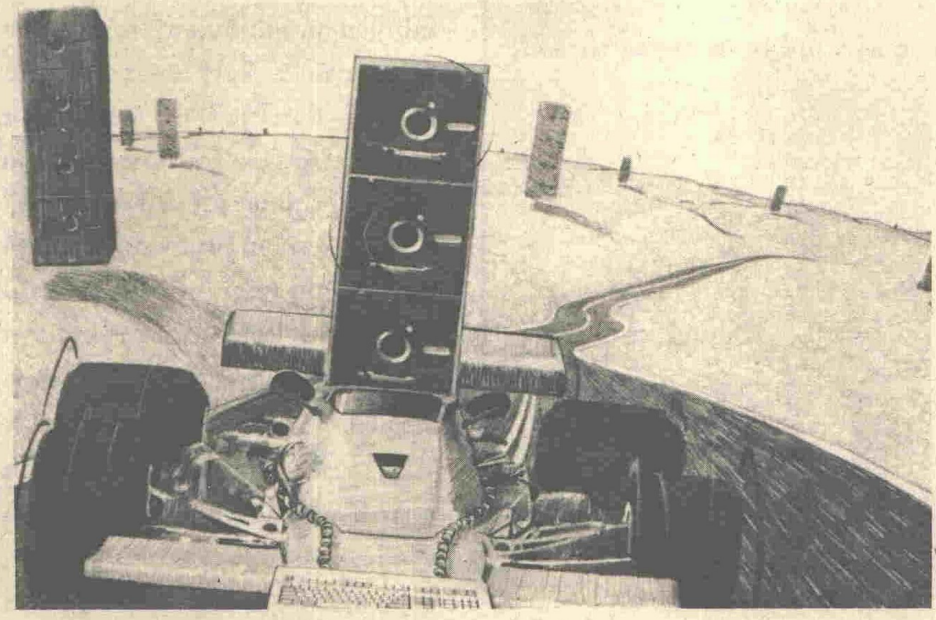
A=0, B=1 สามารถเขียนโพรซีเดอร์ ReadSector ได้ดังรูปที่ 5 สำหรับตัวแปรบัฟเฟอร์ซึ่งใช้เป็นพอยน์เตอร์ของตัวอักษร เพื่อให้สามารถอ่านจำนวนเซกเตอร์ตาม nsctr ลงบนตัวแปรขนาดใด ๆ ก็ได้ โดยส่งค่ามาเฉพาะพื้นที่ (address) แรกของตัวแปรที่ต้องการเก็บลงดิสค์ ตัวอย่างโปรแกรมที่เขียนในบทความนี้ทั้งหมดจะไม่ตรวจสอบประเภทของข้อผิดพลาดในการทำงาน และจะหยุดโปรแกรมทันทีเมื่อมีข้อผิดพลาดในการอ่านดิสค์ ถ้าผู้ใช้ต้องการตรวจสอบข้อผิดพลาดสามารถเขียนเพิ่มเติมจากค่าในรีจิสเตอร์ AH ดังรูปที่ 6 ค่าของ AH จะเก็บรหัสผิดพลาดของการทำงานไว้ ในการอ่านข้อมูลที่ตำแหน่งของตัวแปร Buffer^ จะใช้ฟังก์ชัน ReadMem ในรูปที่ 7 ซึ่งจะรับค่า indx เป็นตำแหน่งใน Buffer^ ที่ต้องการอ่านค่าเริ่มจากตำแหน่งที่ 1 การแสดงข้อมูลใน Buffer^ ที่อ่านมาจะใช้โพรซีเดอร์ DisplaySector ในรูปที่ 8 การแสดงข้อมูลจะแบ่งเป็นแถวละ 16

```
26: Function ReadMem(Buffer:BufferType;indx:word):byte;
27: var  segment,offset : word;
28: begin
29:    segment:=Seg(Buffer^);
30:    offset:=Ofs(Buffer^);
31:    ReadMem:=mem[segment:offset-1+indx];
32: end;
33:
34: Function HexStr(num:byte):string;
35: const HexDigit : string[16] = '0123456789ABCDEF';
36: var   i,mask    : integer;
37:       temps     : string;
38: begin
39:    temps:='';
40:    for i:=1 to sizeof(num)*2 do begin
41:       mask:=num and $000F;
42:       num:=num shr 4;
43:       temps:=HexDigit[mask+1]+temps;
44:    end;
45:    HexStr:=temps;
46: end;
47:
```

รูปที่ 7 ฟังก์ชัน ReadMem

```
48: Procedure ClearDisplay;
49: begin
50:    Window(1,6,80,25);
51:    ClrScr;
52:    Window(1,1,80,25);
53: end;
54:
55: Procedure DisplaySector(Buffer:BufferType);
56: var loopi,loopj     : integer;
57:     bdata           : byte;
58: begin
59:    Window(1,10,80,25);
60:    ClrScr;
61:    for loopi:=1 to 32 do begin
62:       Write(loopi:2,': ');
63:       for loopj:=1 to 16 do begin
64:          bdata:=ReadMem(Buffer,(loopi-1)*16+loopj);
65:          Write(HexStr(bdata):3);
66:       end;
67:       Write('   ');
68:       for loopj:=1 to 16 do begin
69:          bdata:=ReadMem(Buffer,(loopi-1)*16+loopj);
70:          if (bdata>=32) and (bdata<=127) then
71:             Write(char(bdata))
72:          else
73:             Write('.');
74:       end;
75:       if loopi mod 16 = 0 then begin
76:          ch:=ReadKey;
77:          ClrScr;
78:       end else
79:          WriteLn;
80:    end;
81:    ClearDisplay;
82: end;
83:
```

รูปที่ 8 โพรซีเดอร์ ClearDisplay



```
84: Procedure ShowSector;
85: var   drive,track,sector,side : integer;
86:       Buffer                  : BufferType;
87: begin
88:    GetMem(Buffer,512);
89:    GotoXY(1,6);
90:    Write('DRIVE  :');
91:    Readln(drive);
92:    Write('TRACK  :');
93:    Readln(track);
94:    Write('SECTOR :');
95:    Readln(sector);
96:    Write('SIDE   :');
97:    Readln(side);
98:    ReadSector(drive,track,sector,side,1,Buffer);
99:    DisplaySector(Buffer);
100:   FreeMem(Buffer,512);
101: end;
102:
```

รูปที่ 9 โพรซีเดอร์ ShowSector

ไบต์ โดยแสดงเป็นเลขฐานสิบหกทางซ้ายมือและแสดงเป็นตัวอักษรทางขวามือ เพราะว่าในหนึ่งเซกเตอร์มีข้อมูล 512 ไบต์ ดังนั้นต้องใช้จำนวนแถวเท่ากับ 32 แถวซึ่งมากกว่าหนึ่งหน้าจอ จึงแบ่งการแสดงครั้งละ 16 แถว แล้วรอให้ผู้ใช้กดคีย์จึงแสดงอีก 16 แถวที่เหลือ ข้อมูลที่อ่านได้จากฟังก์ชัน ReadMem สามารถเปลี่ยนเป็นค่าในฐานสิบหกได้ด้วยฟังก์ชัน HexStr หลังจากนั้นก็เรียกใช้โพรซีเดอร์ ClearDisplay เพื่อลบหน้าจอส่วนที่ใช้ในการแสดงข้อมูล โพรซีเดอร์ ReadSector และ DisplaySector จะถูกเรียกใช้โดยโพรซีเดอร์ ShowSector ในรูปที่ 9 การทำงานของโพรซีเดอร์ ShowSector จะเริ่มจากการใช้โพรซีเดอร์ GetMem ทำการจองหน่วยความจำเท่ากับขนาดของข้อมูลในหนึ่งเซกเตอร์คือ 512 ไบต์ แล้วเก็บตำแหน่งของหน่วยความจำที่จองไว้ในตัวแปรบัฟเฟอร์ โดยใช้คำสั่ง ReadLn อ่านค่าตำแหน่งเซกเตอร์บนแผ่นดิสค์ เมื่ออ่านข้อมูลบนแผ่นดิสค์ด้วยโพรซีเดอร์

```
Call with:  AH          = 03H
            AL          = จำนวนเซกเตอร์ที่จะเขียน
            ES:BX       = เซกเมนต์และออฟเซ็ตของ
                          บัฟเฟอร์ที่กำหนด
            CH          = หมายเลขเทร็ก (0-39)
            CL          = หมายเลขเซกเตอร์ (1-9)
            DH          = หมายเลขหน้าของดิสค์ (0-9)
            DL          = หมายเลขไดร์ฟ (0-3)
Return:  ถ้าการเขียนสำเร็จ
            Carry flag  = clear
            AH          = 0
            AL          = จำนวนเซกเตอร์ที่เขียนได้
         ถ้าการอ่านล้มเหลว
            Carry flag  = set
            AH          = บอกสถานะของข้อผิดพลาดถูกเซ็ตเมื่อ
                  บิท 0 = illegal command passed to drive
                  บิท 1 = disk write-protected
                  บิท 2 = requested sector not found
                  บิท 3 = DMA overrun on operation
                  บิท 4 = data error on disk read (CRC)
                  บิท 5 = controller error
                  บิท 6 = seek failure
                  บิท 7 = disk timed-out (failed to respond)
```

รูปที่ 10 แสดงการใช้อินเตอร์รัพต์หมายเลข $13 ฟังก์ชัน 3

```
103: Procedure WriteSector(drv,trck,sctr,sd,nsctr:integer;Buffer:BufferType);
104: var reg : Registers;
105: begin
106:    reg.ah:=03;
107:    reg.al:=nsctr;
108:    reg.es:=Seg(Buffer^);
109:    reg.bx:=Ofs(Buffer^);
110:    reg.ch:=trck;
111:    reg.cl:=sctr;
112:    reg.dh:=sd;
113:    reg.dl:=drv;
114:    intr($13,reg);
115:    if reg.ah<>0 then begin
116:       WriteLn;
117:       WriteLn('ERROR CODE =',reg.ah);
118:       Halt(1);
119:    end;
120: end;
121:
```

รูปที่ 11 โพรซีเดอร์ WriteSector

```
122: Procedure DiskCopy_Track;
123: var loop_trck,loop_sd  : integer;
124:     Buffer            : BufferType;
125: begin
126:    GetMem(Buffer,4608);
127:    GotoXY(10,10);
128:    Write('TRACK:       SIDE:');
129:    for loop_trck:=0 to 39 do begin
130:       GotoXY(16,10);
131:       Write(loop_trck);
132:       for loop_sd:=0 to 1 do begin
133:          GotoXY(28,10);
134:          Write(loop_sd);
135:          ReadSector(0,loop_trck,1,loop_sd,9,Buffer);
136:          WriteSector(1,loop_trck,1,loop_sd,9,Buffer);
137:       end;
138:    end;
139:    FreeMem(Buffer,4608);
140:    GotoXY(10,11);
141:    Write('Copy Complete');
142:    ch:=ReadKey;
143:    ClearDisplay;
144: end;
145:
```

รูปที่ 12 โพรซีเดอร์ DiskCopy-Track

ReadSector แล้วจึงนำมาแสดงด้วยโพรซีเดอร์ DisplaySector

เขียนข้อมูลในเซกเตอร์

ส่วนการเขียนข้อมูลลงบนแผ่นดิสค์จะใช้อินเตอร์รัพต์หมายเลข $13 ฟังก์ชัน 3 แสดงรายละเอียดในรูปที่ 10 มีลักษณะคล้ายกับอินเตอร์รัพต์หมายเลข $13 ฟังก์ชัน 2 ซึ่งใช้ในการอ่านข้อมูลบนดิสค์ ส่วนโพรซีเดอร์ WriteSector แสดงในรูปที่ 11 จะใช้ในการอ่านข้อมูลจากดิสค์ ในขณะนี้เรามีทั้งโพรซีเดอร์ที่ใช้อ่านและเขียนข้อมูลลงบนแผ่นดิสค์แล้ว ต่อไปเราจะนำไปใช้ในการก๊อบปี้ดิสค์ทั้งแผ่น โดยทำการอ่านข้อมูลจากดิสค์ครั้งละ 1 แทร็กซึ่งมีขนาด 9 เซกเตอร์ แล้วทำการก๊อบปี้ข้อมูลจากไดร์ฟ A ไปยังไดร์ฟ B ครั้งละ 1 แทร็ก เริ่มจากแทร็กที่ 0 ไปจนกระทั่งครบ 39 ดังนั้นจึงต้องจองเนื้อที่สำหรับเก็บข้อมูลเท่ากับ 9 คูณ 512 เท่ากับ 4608 ไบต์ ดังรูปที่ 12

```
146: Procedure DiskCopy_Disk;
147: var   Buffer           : array[0..39] of array[0..1] of BufferType;
148:       loop_trck,loop_sd : integer;
149: begin
150:    for loop_trck:=0 to 39 do
151:       for loop_sd:=0 to 1 do
152:          GetMem(Buffer[loop_trck][loop_sd],4608);
153:    GotoXY(10,10);
154:    Write('READING TRACK:');
155:    for loop_trck:=0 to 39 do begin
156:       GotoXY(24,10);
157:       Write(loop_trck);
158:       for loop_sd:=0 to 1 do
159:          ReadSector(0,loop_trck,1,loop_sd,9,Buffer[loop_trck][loop_sd]);
160:    end;
161:    GotoXY(10,11);
162:    Write('WRITING TRACK:');
163:    for loop_trck:=0 to 39 do begin
164:       GotoXY(24,11);
165:       Write(loop_trck);
166:       for loop_sd:=0 to 1 do
167:          WriteSector(1,loop_trck,1,loop_sd,9,Buffer[loop_trck][loop_sd]);
168:    end;
169:    for loop_trck:=0 to 39 do
170:       for loop_sd:=0 to 1 do
171:          FreeMem(Buffer[loop_trck][loop_sd],4608);
172:    GotoXY(10,12);
173:    Write('Copy Complete');
174:    ch:=ReadKey;
175:    ClearDisplay;
176: end;
177:
```

รูปที่ 13 โพรซีเดอร์ DiskCopy-Disk

	เซกเตอร์สัมบูรณ์เริ่มต้น	จำนวนเซกเตอร์
ดอสบูตเรคอร์ด	0	1
FAT	1	2
FAT สำรอง	3	2
รากไดเรคทอรี	5	7
เซกเตอร์เก็บข้อมูล	12	708

ตารางที่ 2 แสดงพื้นที่บนดิสก์

(แผ่นดิสก์มีสองหน้า ดังนั้นจึงมีตัวแปร loop-sd สำหรับบอกความหมายเลขหน้าจากหน้า 0 ถึงหน้า 1)

จะเห็นว่าเราสามารถก๊อบปี้ข้อมูลครั้งละแทร็กได้ถ้าเราจะอ่านข้อมูลทั้งแผ่นรอบเดียวแล้วค่อยนำมาเขียนลงบนแผ่นดิสก์อีกแผ่น จะสามารถทำงานได้ดีกว่าเดิม ดังนั้นการสร้างโพรซีเดอร์ DiskCopy_Disk ในรูปที่ 13 จึงต้องมีการสร้างตัวแปรบัฟเฟอร์ให้สามารถเก็บข้อมูลได้ทั้งแผ่น จึงต้องใช้หน่วยความจำมากขึ้น ดังนั้นการอ่านข้อมูลจะกระทำครั้งละแทร็กตามเดิม แต่จะอ่านหมดทุกแทร็กในไดรฟ์ A แล้วก๊อบปี้ลงบนไดรฟ์ B ทีละแทร็กจนครบ ในการทำงานของโพรซีเดอร์ DiskCopy_Disk นี้ไม่สามารถทำงานใน IDE (Integrated Development Environment) ของเทอร์โบปาสคาลได้ เนื่องจากต้องจองหน่วยความจำเท่ากับดิสก์หนึ่งแผ่น (368640 ไบต์) ถ้าต้องการทดลองการทำงานควรคอมพายล์เป็น EXE ก่อนแล้วค่อยเรียกการทำงานจากดอส

เบื้องหลังบูตเรคอร์ด

การก๊อบปี้แผ่นดิสก์ทั้งแผ่นที่กล่าวมานั้น เราไม่สนใจว่าลักษณะการจัดเรียงข้อมูลของดิสก์เป็นอย่างไร ดังนั้นในหัวข้อนี้เราจะพิจารณาการจัดพื้นที่บนแผ่นดิสก์ว่าใช้เซกเตอร์ไหนทำอะไรบ้าง ตารางที่ 2 จะแสดงพื้นที่บนแผ่นดิสก์ที่ใช้ เริ่มจากดอสบูตเรคอร์ด (DOS boot record) ที่ตำแหน่ง (0,0,1) มีขนาด 1 เซกเตอร์ ซึ่งมีรายละเอียดของข้อมูลและขนาดของข้อมูลที่เก็บไว้ในบูตเรคอร์ดดังนี้

คำสั่ง JMP	3 ไบต์
ชื่อของดอสเป็น IBM หรือ MS	8 ไบต์
ขนาดของข้อมูลในหนึ่งเซกเตอร์	2 ไบต์
จำนวนของเซกเตอร์ต่อคลัสเตอร์	1 ไบต์
เซกเตอร์สงวน	1 ไบต์
จำนวนชุดของ FAT	1 ไบต์
จำนวนไฟล์สูงสุดที่ระดับราก	2 ไบต์
จำนวนเซกเตอร์ทั้งแผ่น	2 ไบต์
ประเภทของดิสก์	1 ไบต์
จำนวนเซกเตอร์ต่อ FAT	2 ไบต์
จำนวนหน้าของดิสก์	1 ไบต์
จำนวนเซกเตอร์ที่ไม่ใช้	2 ไบต์
โปรแกรมบูตสแตร็ป	ที่เหลือ

โพรซีเดอร์ BootRecord ในรูปที่ 14 จะอ่านข้อมูลจากบูตเรคอร์ดขึ้นมาเพื่อแสดงรายละเอียดบนหน้าจอ คลัสเตอร์เป็นกลุ่มของเซกเตอร์ที่ดอสใช้ในการจัดเก็บข้อมูล ดิสก์ประเภทต่าง ๆ มี

```
178: Procedure BootRecord;
179: var  Buffer              : BufferType;
180:      name                : string[8];
181:      loop,BytePerSector  : word;
182: begin
183:    GetMem(Buffer,512);
184:    ReadSector(1,0,1,0,1,Buffer);
185:    Window(1,10,80,25);
186:    for loop:=1 to 8 do
187:       name[loop]:=chr(ReadMem(Buffer,3+loop));
188:    name[0]:=#8;
189:    WriteLn('Systen name          :',name);
190:    WriteLn('Bytes per sector     :',
191:            ReadMem(Buffer,13) shl 8+ReadMem(Buffer,12));
192:    WriteLn('Sectors per cluster  :',ReadMem(Buffer,14));
193:    WriteLn('Reserved sectors     :',
194:            ReadMem(Buffer,16) shl 8+ReadMem(Buffer,15));
195:    WriteLn('Number of FAT copies :',ReadMem(Buffer,17));
196:    WriteLn('Max root directory   :',
197:            ReadMem(Buffer,19) shl 8+ReadMem(Buffer,18));
198:    WriteLn('Total sectors        :',
199:            ReadMem(Buffer,21) shl 8+ReadMem(Buffer,20));
200:    WriteLn('Media description    :','$',HexStr(ReadMem(Buffer,22)));
201:    WriteLn('Sectors per FAT      :',
202:            ReadMem(Buffer,24) shl 8+ReadMem(Buffer,23));
203:    WriteLn('Sectors per track    :',
204:            ReadMem(Buffer,26) shl 8+ReadMem(Buffer,25));
205:    WriteLn('Sides per disk       :',
206:            ReadMem(Buffer,28) shl 8+ReadMem(Buffer,27));
207:    WriteLn('Hidden sector        :',
208:            ReadMem(Buffer,30) shl 8+ReadMem(Buffer,29));
209:    FreeMem(Buffer,512);
210:    ch:=ReadKey;
211:    ClearDisplay;
212: end;
213:
```

รูปที่ 14 โพรซีเดอร์ BootRecord

จำนวนเซกเตอร์ต่อคลัสเตอร์ต่างกัน สำหรับดิสค์ที่ใช้ทั่วไปมีจำนวน 2 เซกเตอร์ต่อคลัสเตอร์ จำนวนชุดของ FAT (File Allocation Table) หมายถึงจำนวน FAT ที่ใช้รวมกับ FAT สำรองว่ามีทั้งหมดกี่ชุด ดิสค์ทั่วไปมีจำนวนชุดของ FAT เท่ากับสองเป็น FAT จริงที่ใช้ชุดหนึ่งและชุดสำรองอีกชุดหนึ่ง สำหรับประเภทของดิสค์จะบอกเป็นโค้ดซึ่งมี

$F8 เป็นฟิกซ์ดิสค์
$F9 เป็นดิสค์ 2 หน้า ขนาด 5 1/4 นิ้ว มี 15 เซกเตอร์ต่อแทร็ก
$FC เป็นดิสค์ 1 หน้า ขนาด 5 1/4 นิ้ว มี 9 เซกเตอร์ต่อแทร็ก
$FD เป็นดิสค์ 2 หน้า ขนาด 5 1/4 นิ้ว มี 9 เซกเตอร์ต่อแทร็ก
$FE เป็นดิสค์ 1 หน้า ขนาด 5 1/4 นิ้ว มี 8 เซกเตอร์ต่อแทร็ก
$FF เป็นดิสค์ 2 หน้า ขนาด 5 1/4นิ้ว มี 8 เซกเตอร์ต่อแทร็ก

สำหรับดิสค์ที่ใช้ทั่วไปมีค่าของประเภทดิสค์เป็น $FD หมายถึงดิสค์สองหน้าขนาด 5 1/4 นิ้ว มีจำนวน 9 เซกเตอร์ต่อแทร็ก

ความหมายของ FAT

FAT จะอยู่ถัดจากบูตเรคอร์ด มีขนาดสองเซกเตอร์ เริ่มที่ตำแหน่ง (0,0,2) จนถึงตำแหน่ง (0,0,3) ต่อจากนั้นก็จะเป็น FAT สำรองมีขนาดสองเซกเตอร์เช่นกัน เริ่มจากตำแหน่ง (0,0,4) ถึงตำแหน่ง (0,0,5) โดยปกติข้อมูลใน FAT ทั้งสองชุดจะเหมือนกัน ในไบต์แรกจะบอกประเภทของดิสค์ ส่วนไบต์ที่ 2 และ 3 จะมีค่าเป็น $FF ตั้งแต่ไบต์ที่ 4 ขึ้นไปจะเป็นโค้ดแสดงสถานะของแต่ละคลัสเตอร์บนแผ่นดิสค์ ซึ่งใช้ข้อมูลขนาด 12 บิตแสดงสถานะของคลัสเตอร์ 1 คลัสเตอร์ โดยเริ่มจากโค้ดของคลัสเตอร์ที่ 2 ในตำแหน่งไบต์ที่ 4 การหาโค้ดแสดงสถานะของคลัสเตอร์เริ่มจากการนำหมายเลขของคลัสเตอร์ที่ต้องการมาคูณด้วย 1.5 แล้วปัดเศษทิ้งกำหนดค่าเป็น i อ่านค่าตัวเลขขนาดสองไบต์จากตำแหน่งที่ i ได้ค่าเป็นโค้ด ถ้าหมายเลขคลัสเตอร์ที่ต้องการเป็นเลขคี่ให้ใช้ค่าโค้ดเท่ากับ code shr 4 แต่ถ้าหมายเลขของคลัสเตอร์เป็นเลขคู่ค่าโค้ดจะเท่ากับ code and $0FFF ค่าของโค้ดที่ได้ท้ายสุดมีความหมายดังนี้

0 : คลัสเตอร์ว่างสามารถนำไปใช้งานได้
2-2371 : เป็นตัวเลขแสดงหมายเลขคลัสเตอร์ที่เก็บข้อมูลของไฟล์ไว้บนดิสค์ สำหรับดิสค์ที่ใช้กันอยู่ทั่วไปหมายเลขคลัสเตอร์เริ่มต้นที่ 2 และคลัสเตอร์สูงสุดคือ 355
$FFF : หมายความว่าคลัสเตอร์นี้เป็นคลัสเตอร์สุดท้ายของไฟล์
$FF7 : เครื่องหมายแสดงถึงคลัสเตอร์ที่เสียไม่สามารถนำไปใช้งาน

ฟังก์ชัน ClusterCode ในรูปที่ 15 จะทำหน้าที่แปลงข้อมูลใน FAT โดยมีตัวแปรบัฟเฟอร์และใช้ clstr เป็นตัวบอกหมายเลขคลัสเตอร์เพื่ออ่านโค้ดที่อยู่ใน FAT สำหรับโพรซีเดอร์ CheckDisk ในรูปที่ 16 จะทำการค้นหาจำนวนคลัสเตอร์ที่เสียออกมาแสดง โดยการเรียกใช้ฟังก์ชัน ClusterCode เพื่อหาคลัสเตอร์ที่มีโค้ดใน FAT เท่ากับ $FF7

```
214: Function ClusterCode(Buffer:BufferType;clstr:integer):integer;
215: var  i,code : integer;
216: begin
217:    i:=Trunc(clstr*1.5);
218:    code:=ReadMem(Buffer,i+2) shl 8+ReadMem(Buffer,i+1);
219:    if clstr mod 2=1 then
220:       code:=code shr 4
221:    else
222:       code:=code and $0FFF;
223:    ClusterCode:=code;
224: end;
225:
```

รูปที่ 15 ฟังก์ชัน ClustrCode

เก็บไดเรคทอรีอย่างไร

ถัดจาก FAT สำรองจะเป็นเซกเตอร์ที่เก็บรายชื่อของไฟล์อยู่ในรากที่เรียกว่าไดเรคทอรี (Directory) จำนวนของเซกเตอร์ที่ใช้เก็บไดเรคทอรีของดิสก์แต่ละประเภทนั้นได้แสดงไว้ในตารางที่ 3

สำหรับดิสก์สองหน้าใช้เซกเตอร์สำหรับเก็บไดเรคทอรี 7 เซกเตอร์ โดยเริ่มจากตำแหน่ง (0,0,6) จนถึงตำแหน่ง (0,0,9) จำนวน 4 เซกเตอร์ แล้วต่อไปยังตำแหน่ง (0,1,1) ถึงตำแหน่ง (0,1,3) อีก 3 เซกเตอร์ รวมเป็น 7 เซกเตอร์ รายชื่อของแต่ละไฟล์ในไดเรคทอรีจะใช้ข้อมูลขนาด 32 ไบต์ในการเก็บข้อมูลต่าง ๆ ดังนี้

```
226: Procedure CheckDisk;
227: var   Buffer              : BufferType;
228:       loop_clstr          : integer;
229:       BadCluster          : longint;
230: begin
231:    GetMem(Buffer,1024);
232:    ReadSector(1,0,4,0,2,Buffer);
233:    BadCluster:=0;
234:    for loop_clstr:=2 to 355 do begin
235:       if ClusterCode(Buffer,loop_clstr)=$FF7 then
236:          Inc(BadCluster,1);
237:    end;
238:    FreeMem(Buffer,1024);
239:    GotoXY(10,10);
240:    Write('NUMBER OF BAD CLUSTER : ',BadCluster);
241:    ch:=ReadKey;
242:    GotoXY(10,10);
243:    ClrEol;
244: end;
245:
```

รูปที่ 16 โพรซีเดอร์ CheckDisk

ประเภท	จำนวนเซกเตอร์	จำนวนไฟล์สูงสุดในราก
ดิสก์แบบหน้าเดียว	4	64
ดิสก์แบบสองหน้า	7	112
ความหนาแน่น 4 เท่า	14	224
Fixed ดิสก์	32	64

ตารางที่ 3 แสดงจำนวนเซกเตอร์ที่ใช้เก็บไดเรคทอรีของดิสก์แต่ละประเภท

ชื่อไฟล์	8 ไบต์
นามสกุลไฟล์	3 ไบต์
แอตทริบิวต์บอกลักษณะไฟล์	1 ไบต์
สงวนสำหรับดอส	10 ไบต์
เวลาและวันที่	4 ไบต์
คลัสเตอร์แรกของไฟล์	2 ไบต์
ขนาดของไฟล์เป็นไบต์	4 ไบต์

แอตทริบิวต์บอกลักษณะของไฟล์มีความหมายดังนี้

บิต 0	ถูกเซ็ตเมื่อเป็น ReadOnly File
บิต 1	ถูกเซ็ตเมื่อเป็น Hidden File
บิต 2	ถูกเซ็ตเมื่อเป็น System File
บิต 3	ถูกเซ็ตเมื่อเป็น Volume label
บิต 4	ถูกเซ็ตเมื่อเป็น Directory
บิต 5	ถูกเซ็ตเมื่อเป็น Archive File

ส่วนเวลาและวันที่ของไฟล์จะเก็บไว้ในลักษณะที่เรียกว่าบิตฟิลด์ (bit-field) มีขนาด 4 ไบต์โดยแต่ละบิตได้จัดเก็บเรียงลำดับจากบิตต่ำสุดดังนี้

วินาที 5 บิต	นาที 6 บิต	ชั่วโมง 5 บิต
วัน 5 บิต	เดือน 4 บิต	ปี 7 บิต

```
246: Procedure UnpackDateTime(pack:longint;var date,time:string);
247: var  unpack          : DateTime;
248:      i               : integer;
249:      s1,s2,s3,s4,s5  : string;
250: begin
251:    UnpackTime(pack,unpack);
252:    with unpack do begin
253:       str(Year:2,s1);
254:       str(Month:2,s2);
255:       str(Day:2,s3);
256:       str(Hour:2,s4);
257:       str(Min:2,s5);
258:    end;
259:    date:=s2+'-'+s3+'-'+s1;
260:    for i:=1 to length(date) do
261:       if date[i]=' ' then date[i]:='0';
262:    time:=s4+':'+s5;
263:    for i:=1 to length(time) do
264:       if time[i]=' ' then time[i]:='0';
265: end;
266:
```

รูปที่ 17 โพรซีเดอร์ UnpackDateTime

เราสามารถใช้โพรซีเดอร์ UnpackTime ของเทอร์โบปาสคาลแปลงความหมายเป็นวันและเวลาใส่ในตัวแปรได้ ดังโพรซีเดอร์ UnpackDateTime ในรูปที่ 17 สำหรับชื่อไฟล์ที่อ่านได้ดอสจะนำชื่อไฟล์ตัวแรกมาพิจารณาก่อน ถ้าหากมีค่าเท่ากับแอสกี 229 แสดงว่าไฟล์นั้นถูกลบที่ไปแล้ว ซึ่งการล้างไฟล์ของดอสจะใช้วิธีเปลี่ยนอักษรตัวแรกของชื่อไฟล์ และรหัสใน FAT แต่จะไม่ทำลายข้อมูลที่อยู่ในคลัสเตอร์ ถ้าชื่อไฟล์ตัวแรกมีค่าแอสกีเท่ากับ 0 หมายถึงสิ้นสุดไฟล์ในไดเรคทอรี ตัวอย่างของไดเรคทอรีได้แสดงไว้ในรูปที่ 18 ด้วยโพรซีเดอร์ Directory ซึ่งจะแสดงเฉพาะ

```
267: Procedure Directory;
268: type  FileType  = record
269:           name        : array[1..8] of byte;
270:           extension   : array[1..3] of byte;
271:           attribute   : byte;
272:           reserved    : array[1..10] of byte;
273:           timepack    : longint;
274:           start       : word;
275:           size        : longint;
276:       end;
277: var   Buffer                      : array[1..112] of FileType;
278:       filedata                    : FileType;
279:       i,j,no                      : integer;
280:       fname,fextension,date,time  : string;
281: begin
282:    GotoXY(1,8);
283:    Write('NO.  FILE      EXT  ATTR  DATE        TIME   START    SIZE');
284:    ReadSector(1,0,6,0,4,@Buffer[1]);
285:    ReadSector(1,0,1,1,3,@Buffer[65]);
286:    Window(1,10,80,25);
287:    i:=1;
288:    no:=1;
289:    filedata:=Buffer[i];
290:    while (i<=112) and (filedata.name[1]<>0) do begin
291:       if filedata.name[1]<>229 then begin
292:          fname:='';
293:          fextension:='';
294:          with filedata do begin
295:             for j:=1 to 8 do
296:                fname:=fname+chr(name[j]);
297:             for j:=1 to 3 do
298:                fextension:=fextension+chr(extension[j]);
299:             UnpackDateTime(timepack,Date,time);
300:             Write(no:3,fname:10,fextension:5,attribute:5);
301:             Write(date:13,time:7,start:6,size:9);
302:          end;
303:          if i mod 16=0 then begin
304:             ch:=ReadKey;
305:             ClrScr;
306:          end else
307:             WriteLn;
308:          Inc(no);
309:       end;
310:       Inc(i);
311:       filedata:=Buffer[i];
312:    end;
313:    ch:=ReadKey;
314:    ClearDisplay;
315: end;
316:
```

รูปที่ 18 โพรซีเดอร์ Directory

```
317: Procedure ListChain;
318: type  FileType  = record
319:           name       : array[1..8] of byte;
320:           extension  : array[1..3] of byte;
321:           attribute  : byte;
322:           reserved   : array[1..10] of byte;
323:           timepack   : longint;
324:           start      : word;
325:           size       : longint;
326:         end;
327: var   Dir                : array[1..112] of FileType;
328:       FAT                : BufferType;
329:       filedata           : FileType;
330:       FileNo,no,i,clstr  : integer;
331: begin
332:    GetMem(FAT,1024);
333:    GotoXY(1,6);
334:    Write('FILE NO FOR READ:');
335:    ReadLn(FileNo);
336:    ReadSector(1,0,6,0,4,@Dir[1]);
337:    ReadSector(1,0,1,1,3,@Dir[65]);
338:    ReadSector(1,0,2,0,2,FAT);
339:    no:=0;
340:    i:=0;
341:    repeat
342:       Inc(i);
343:       filedata:=Dir[i];
344:       if filedata.name[1]<>229 then
345:          Inc(no);
346:    until no=FileNo;
347:    Write('FILENAME        :');
348:    i:=1;
349:    repeat
350:       Write(Chr(filedata.name[i]));
351:       Inc(i);
352:    until (i>8) or (filedata.name[i]=32);
353:    Write('.');
354:    for i:=1 to 3 do
355:       Write(Chr(filedata.extension[i]));
356:    Window(1,10,80,25);
357:    clstr:=filedata.start;
358:    repeat
359:       Write(clstr:5);
360:       clstr:=ClusterCode(FAT,clstr);
361:    until clstr=$0FFF;
362:    FreeMem(FAT,1024);
363:    ch:=ReadKey;
364:    ClearDisplay;
365: end;
366:
```

รูปที่ 19 โพรซีเดอร์ ListChain

ไฟล์ที่ยังไม่ถูกลบเท่านั้น เราสามารถหาข้อมูลของไฟล์ที่อ่านจากไดเรคทอรีว่าอยู่ในคลัสเตอร์ไหนโดยการอ่านค่ารหัสของคลัสเตอร์ใน FAT เริ่มจากคลัสเตอร์เริ่มต้นของไฟล์ แล้วอ่านรหัสของคลัสเตอร์ต่อไปจนกว่าจะได้ค่ารหัสเท่ากับ $FFF ซึ่งหมายถึงคลัสเตอร์สุดท้ายของไฟล์ โพรซีเดอร์ ListChain ในรูปที่ 19 จะแสดงหมายเลขคลัสเตอร์ที่เก็บข้อมูล เมื่อเราได้หมายเลขคลัสเตอร์แล้วเราย่อมจะหาหมายเลขเซกเตอร์, แทร็ก และหน้าของแผ่นดิสค์ที่ต้องการได้ ในรูปที่ 20 เป็นส่วนโปรแกรมหลักสำหรับเรียกใช้โพรซีเดอร์ต่าง ๆ ที่กล่าวมาแล้ว จะเห็นว่าเมื่อเข้าสู่โปรแกรมแล้วจะเรียกใช้โพรซีเดอร์ ResetDisk ก่อน ซึ่งเป็นการรีเซตค่าของดิสค์คอนโทรลเลอร์โดยใช้อินเตอร์รัพต์ $13 ฟังก์ชัน 0 โดยไม่ต้องส่งค่าอื่น

```
367: Procedure ResetDisk;
368: var  reg : Registers;
369: begin
370:    reg.ah:=0;
371:    Intr($13,reg);
372: end;
373:
374: BEGIN
375:    ResetDisk;
376:    ClrScr;
377:    WriteLn('[0]  QUIT                          [4] BOOT RECORD B:');
378:    WriteLn('[1]  SHOW SECTOR                   [5] CHECK DISK B:');
379:    WriteLn('[2]  DISKCOPY A: B: [TRACK]        [6] DIRECTORY B: ');
380:    WriteLn('[3]  DISKCOPY A: B: [DISK]         [7] LIST CHAIN   ');
381:    repeat
382:       ch:=ReadKey;
383:       case ch of
384:          '1' : ShowSector;
385:          '2' : DiskCopy_Track;
386:          '3' : DiskCopy_Disk;
387:          '4' : BootRecord;
388:          '5' : CheckDisk;
389:          '6' : Directory;
390:          '7' : ListChain;
391:       end;
392:    until ch='0';
393:    ClrScr;
394: END.
```

รูปที่ 20 โพรซีเดอร์ ResetDisk

ทำอย่างไรถ้าไฟล์ถูกลบ

ดังได้กล่าวมาแล้วว่าในการล้างไฟล์ของดอสนั้น ดอสจะเปลี่ยนชื่อตัวแรกของไฟล์เป็นแอสกี 229 และเปลี่ยนค่าโค้ดใน FAT ของคลัสเตอร์ที่เคยใช้ให้เป็น เพื่อให้ไฟล์อื่นสามารถใช้ได้เมื่อมีการเก็บข้อมูลบนแผ่นดิสค์เพิ่มขึ้น และเมื่อมีการเก็บข้อมูลเนื้อที่ในส่วนดังกล่าวจะถูกใช้ก่อน เนื่องจากดอสจะหาเนื้อที่ของหมายเลขคลัสเตอร์จากน้อยไปมาก ดังนั้นการกู้ข้อมูลของไฟล์ที่โดนลบทิ้งแล้วสามารถทำได้ด้วยการอ่านข้อมูลขึ้นมาครั้งละหนึ่งคลัสเตอร์ เริ่มจากคลัสเตอร์แรกไปยังคลัสเตอร์สุดท้ายของแผ่นดิสค์ โดยจะแสดงข้อมูลบนหน้าจอที่ละคลัสเตอร์ให้ตรวจสอบว่าเป็นส่วนของข้อมูลที่ต้องการกู้หรือไม่ ถ้าใช่จะเก็บข้อมูลนั้นไว้ในตัวแปร ทำเช่นนี้จนหมดแผ่นดิสค์หรือจนกว่าได้ข้อมูลครบ แล้วทำการเขียนข้อมูลลงบนดิสค์โดยใช้คำสั่งเปิดไฟล์เพื่อเขียนใหม่ด้วยคำสั่ง Write จะเห็นว่าการกู้ข้อมูลบนแผ่นดิสค์มีความยุ่งยากพอสมควร ทั้งนี้ต้องพิจารณาความสำคัญของข้อมูลว่ามีความสำคัญมากน้อยเพียงใด อย่างไรก็ตาม ในปัจจุบันมีโปรแกรมสำเร็จรูปที่สามารถใช้ในการกู้ข้อมูลออกวางจำหน่ายในท้องตลาดมากมาย ดังนั้นถ้าผู้อ่านเข้าใจลักษณะโครงสร้างของดิสค์แล้วการใช้โปรแกรมกู้ข้อมูลต่าง ๆ จะได้ผลและปลอดภัยยิ่งขึ้น

รู้จักกับซับไดเรคทอรี

การเก็บข้อมูลบนแผ่นดิสค์นอกจากจะเก็บเป็นไฟล์แล้วยังสามารถจัดเก็บเป็นซับไดเรคทอรี (Sub Directory) ได้ด้วย การสร้าง, การทำลายซับไดเรคทอรีจะใช้คำสั่ง MKDIR และ RMDIR ตามลำดับ เมื่อเราอ่านชื่อไฟล์และได้ค่าแอตทริบิวต์บอกลักษณะไฟล์ว่าเป็นไดเรคทอรี คือ บิตที่ 4 หมายเลขคลัสเตอร์แรกของไฟล์ ซึ่งปกติใช้ในการเก็บข้อมูลจะถูกนำไปใช้เก็บรายชื่อของไฟล์ในซับไดเรคทอรี จะมีลักษณะการเก็บไฟล์ครั้งละ 32 ไบต์เช่นเดียวกับราก และไฟล์ที่อยู่ในซับไดเรคทอรีก็สามารถเป็นไดเรคทอรีได้ ดังนั้นจะเห็นว่าจำนวนไฟล์ในซับไดเรคทอรีจะมีไม่จำกัดตราบเท่าที่แผ่นดิสค์ยังมีที่ว่างอยู่

สรุป

โปรแกรมยูทิลิตี้ที่ช่วยในการซ่าแหละดิสเกตต์คงจะช่วยให้ผู้ใช้เครื่องไมโครคอมพิวเตอร์ทุกคนได้เข้าถึงลักษณะโครงสร้างในดิสค์และสามารถจะเก็บรักษาข้อมูลได้อย่างปลอดภัย ถ้าข้อมูลถูกลบไปโดยไม่ตั้งใจหรือดิสค์ที่ใช้เก็บข้อมูลอ่านไม่ได้ ผู้ใช้ก็สามารถที่จะกู้ข้อมูลกลับคืนมาด้วยตัวเองได้ แถมยังไม่ต้องเสียเวลามานั่งพิมพ์ข้อมูลเข้าไปใหม่ หวังว่าโปรแกรมนี้คงจะเป็นประโยชน์ต่อผู้อ่านบ้างตามสมควร ถ้าต้องการใช้งานโปรแกรมนี้ติดต่อซอฟต์เมล์ฉบับนี้ได้ที่กองบรรณาธิการ วารสารไมโครฯนะครับ